Australian Government
Department of Defence
Defence Science and
Technology Organisation

An Implementation of Adaptive Side Lobe Cancellation in MATLAB®

A. P. Shaw

Electronic Warfare and Radar Division

Defence Science and Technology Organisation

DSTO–TN–0955

ABSTRACT

This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.

Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone: (08) 7389 5555
Facsimile: (08) 7389 6567

AR No. 014–078
July, 2010

APPROVED FOR PUBLIC RELEASE

An Implementation of Adaptive Side Lobe Cancellation in MATLAB®

Executive Summary

The objective of an adaptive side lobe cancellation system is to suppress high duty cycle and noise-like interference signals received through the side lobes of the radar. This is accomplished by using auxiliary antennas whose antenna pattern approximates to the side lobe pattern of the main antenna. By suitably phasing the signals received by the auxiliaries a directional anti-phase signal can be generated that when added to the main antenna signal "subtracts" the interference.

This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.

Contents

Appendices

1 Introduction

The objective of an adaptive side lobe cancellation (ASLC) system is to suppress high duty cycle and noise-like interference signals received through the side lobes of the radar. This is accomplished by using auxiliary antennas whose antenna pattern approximates to the side lobe pattern of the main antenna. By suitably phasing the signals received by the auxiliaries a directional anti-phase signal can be generated that when added to the main antenna signal "subtracts" the interference.

A side lobe canceller effectively works by forming a beam from the auxiliary antennas that points towards the interference source, the signal from this beam is then subtracted from the signal detected by the main antenna. The problem is how do we determine the direction in which to point the beam we form from the auxiliary antennas.

Mathematically the problem is to estimate the complex vector of N weights $\mathbf{W} = (W_1, W_2, \ \ldots W_N)$ to apply to the N signals $\mathbf{V} = (V_1, V_2, \ldots V_N)$ from the auxiliary antennas in order to minimise up to N directional interference signals in the side lobes of the main antenna signal (V_m). It may be shown [1] that for band limited noise, appropriate weights may be estimated from:

$$\mathbf{W} = \mu \mathbf{M}^{-1} \mathbf{R} \tag{1}$$

$$\text{where} \quad M = E\left\{\mathbf{V}^* \mathbf{V}^T\right\} \tag{2}$$

$$\text{and} \quad R = E\left\{V_m \mathbf{V}^*\right\} \tag{3}$$

where μ is an arbitrary scalar $E\{\ \ \}$ the expected value, * the complex conjugate, ^{T} the transpose and $^{-1}$ the inverse.

2 ASLC inputs

2.1 On nomenclature

The code nomenclature assumes that the data has been Doppler processed and pulse compressed, but that is not essential.

The association of particular matrix dimensions with "range" or "Doppler" is one of convenience ("range" is the second dimension and "Doppler" the third dimension of the signal and auxiliary data) and need not necessarily align with the user's data. However, if there is a change in the association, then it is the responsibility of the user to ensure that the parameter values provided are appropriate to the interpretation of the data.

2.2 Sample data

The signal data against which ASLC is to be applied is the `Signal`, which consists of an arbitrary number of "beams" or "channels" in the form of a three dimensional array such that `(nCs, nRs, nDs)=size(Signal)` where `nCs` is the number of signal channels, `nRs` is the number of range bins and `nDs` is the number of Doppler bins.

The auxiliary data that is to provide the ASLC is `Auxiliaries`, which consists of an arbitrary number of "beams" or "channels" in the form of a three dimensional array such that `(nCa, nRa, nDa)=size(Auxiliaries)` where `nCa` is the number of signal channels, `nRa` is the number of range bins and `nDa` is the number of Doppler bins. It is essential that `nRa == nRs` and `nDa == nDs`.

2.3 Range strides

Simple ASLC is implemented with a single set of training data over the data set. This is not desirable if the interference is not uniform over the data set. The baseline code directly supports the segmentation of the data into "range strides", with calculation of the adaptive weight carried out for each stride. The number of strides used is determined by the `Nstride` parameter. The application of the weights across the data is then by interpolation, so that each range bin has a unique set of weights.

In some circumstances it is desirable to not include data from the range region at the centre of the stride. The size of the excluded region is defined by the `RangeGap` parameter.

2.4 Training region

The extent of the training region is established by the `ClutterExtentIn` vector.

If `length(ClutterExtentIn)==4` then the clutter region is assumed to extend over two regions from `Init_bin=ClutterExtentIn(1)` to `low_bin=ClutterExtentIn(2)` and from `high_bin=ClutterExtentIn(3)` to `final_bin=ClutterExtentIn(4)`.

If `length(ClutterExtentIn)==2` then it is assumed that `Init_bin=1` and `final_bin =nDa`. The values in `ClutterExtentIn` are sorted into an ascending list so that they cannot overlap, although the regions may merge.

The sample region used is determined by the `SampleRegion` parameter.

When `SampleRegion=="noise"` then the training is obtained by a single region (from `low_bin` to `high_bin`) in the data and the ASLC corrections are applied over the entire data space.

When `SampleRegion=="clutter"` then the training is obtained from *two* sample regions (from `Init_bin` to `low_bin` and from `high_bin` to `final_bin`) and the ASLC corrections are applied only over the training region.

Any other value for `SampleRegion` results in a selection of training data on either side of the middle Doppler bin until the required number of samples has been obtained. The resulting ASLC corrections are applied over the entire data space.

2.5 Training data

To train the data, samples from the training region must be selected. The number of samples in range (`Nranges` parameter) and Doppler (`Ndopplers` parameter) to be independently defined.

Radar data may include correlations in range and/or Doppler due to the waveform design, oversampling, compression weights, Doppler padding etc. This can result in the use of correlated data in the training which does not provide useful training data (at least not in the baseline implementation). To overcome this the software supports the ability to require that only every n^{th} data sample is used. The spacing for range space and for Doppler space can differ. If the required number of samples will take the data outside of the region of validity then the software makes appropriate adjustments to the data collected. The spacing between samples is determined by the `Nstep` vector. If `Nstep` is a single value then the value is used for both range and Doppler, if it is an array, then the first value is used for the range dimension and the second value used for the Doppler dimension.

There is *no* checking that the number of samples is sufficient for an accurate estimate of the covariance matrix.

2.6 Pre-computed weights

In some circumstances it is desirable to use pre-computed weights from a prior data set. This is supported by providing in the output from the function the weights that have been estimated. These weights may then be placed as an input to the function in the variable `InWeights`. If this function is not required, then this variable must be set to `InWeights=[]`. The implementation assumes (apart from a few simple checks) that the supplied weights are appropriate to the data set that they are being used on.

3 ASLC outputs

3.1 Output data

The outputs data set is the same format and size as the input signal data, but after ASLC has been applied to the signal.

3.2 Weights data

Weights data that is calculated by the code is supplied as an additional output. This enables examination of the weight magnitudes, but also the use of the weights generated with a similar (same size) signal and auxiliary data (with the other input parameters the same) in accordance with section 2.6.

4 Hidden parameters

The ASLC implementation includes a number of “hard coded” variables that may need to be changed for a particular implementation.

The parameter `ARBITARY_SCALAR` represents μ the arbitrary scalar in equation 1 and is currently set to unity.

In training the data there is a simple test (see source code lines 335-339 and 372-373) for the presence of a large target in the training data (a crude cell average CFAR) so that large targets can be excluded from the training data. The magnitude of target that is considered "large" for exclusion is set by the value of `LARGE_TARGET_THRESHOLD`. In the code herein it is set to 20.

If other processing has been applied to eliminate "zero" Doppler effects such as DPCA which has been trained on the same data as will be used for ASLC, then it is (probably) desirable to exclude that data from the ASLC training data. In the code herein, the region that is excluded is determined by the parameter `DPCA_MASKED` and it ensures that Doppler bins `1:DPCA_MASKED` and bins `nDa-DPCA_MASKED:nDa` are excluded from training.

5 Limitations

There is no conditioning applied in the generation of the covariance matrix in this implementation. Depending upon the data and usage of the code it is possible that the estimate for the covariance matrix will be ill-conditioned for inversion. Generally MATLAB® will throw a warning if this is the case.

The MATLAB® code doesn't make use of the (obvious) `inv(M)*R'` instead it uses the MATLAB® recommendation of `M\R'`. To change to the explicit code the "commenting out" needs to be changed in the source code at lines 390 and 391.

The code has been modified to MATLAB® 2009b. The majority of the code should work with prior versions *except* where the dummy variable `~` has been used in some return values. If the code is to be run on an older implementation of MATLAB® then `~` will need to be changed to (any) convenient (non-clashing) variable name in lines 125 and 129.

References

1. Farina, A. (1990) *Radar Handbook (editor: M Skolnik)*, 2nd edition edn, McGraw-Hill, chapter 9: Electronic-counter-countermeasures, pp. 9.1–9.18.

Appendix A MATLAB implementation of ASLC

The following is the code used to compute and apply the ASLC weights.

```
function [output, weights]=ASLC_ARCS1(Signal, Auxiliaries, Nranges,...
                                      Ndopplers, Nstep, Nstride, ...
                                      SampleRegion, RangeGap, ...
                                      InWeights, ClutterExtentIn)
%function [output, weights]=ASLC_ARCS1(Signal, Auxiliaries, Nranges,...
%                                      Ndopplers, Nstep, Nstride, ...
%                                      SampleRegion, RangeGap, ...
%                                      InWeights, ClutterExtent)
%
% Compute and apply the ASLC weights
%
%Parameters
%  Signal:        [Signal Range Doppler] array for the signal
%                 antenna(s)
%  Auxiliaries:   [Aux Range Doppler] array for the auxiliaries
%  Nranges:       Number of range bins to be used in covariance estimate
%  Ndopplers:     Number of doppler bins to be used in covariance
%                 estimate
%  Nstep:         Number of range & Doppler bins between samples used
%                 in covariance estimate can be a vector for different
%                 step sizes [range Doppler] (not used if InWeights~=[])
%  Nstride:       Number of estimates to be made in range space
%  SampleRegion:  Defines the region within which training data is
%                 collected relative to the ClutterExtentIn values
%                 AND the region to which the processing is applied.
%                 If SampleRegion==``noise'' then the training data is
%                 collected in the region low_bin:high_bin (see
%                 ClutterExtentIn) and is applied to the ENTIRE Doppler
%                 extent of the data. If SampleRegion==``Clutter'' then
%                 training data is collected in two regions
%                 Init_bin:Low_bin and high_bin:final_bin (see
%                 ClutterExtentIn) and the processing is only applied to
%                 the same region. Other values for SampleRegion define
%                 a default space in the middle of the Doppler coverage.
%  RangeGap:      extent of the range gap in around the stride central
%                 range that is to be used in estimating the weights
%                 (not used if InWeights~=[])
%  InWeights:     Precomputed weights, if weights are to be estimated
%                 from the data then this MUST be a null (=[]) array,
%                 if it is NOT null, then the value will be used
%                 (subject to some sanity checks) overwrite Nstride
%                 and as the ASLC weights. The form of InWeights is
%                 assumed to match the form of the output parameter
%                 weights.
%  ClutterExtentIn A Either a four value array [Init_bin low_bin
%                 high_bin final_bin] that defines the clutter region
%                 to be [Init_bin:low_bin] and [high_bin:final_bin],
%                 or a two value array [low_bin high_bin] that
%                 assumes Init_bin=1 and final_bin=nD where nD is the
```

```
50 %                  maximum Doppler bin. Note the final list is always
51 %                  sorted into assending order.
52 %  output:        [Signal Range Doppler] array after ASLC processing
53 %  weights:       [stride Aux] array of the weights.
54 %
55 % Notes:
56 % 1. Code syntax assumes post-Doppler processing.
57 % 2. There is no conditioning of the covariance matrix prior to
58 %    inversion in this code! MATLAB may throw warning messages!
59 %
60 % Original by APShaw, 24 July 2009
61 % Modified for clutter processing (RANGE_GAP)  by APShaw, 10 Aug 2009
62 % Modified for pre-computed weights in clutter by APShaw, 27 Aug 2009
63 % Modified to exclude very edge Doppler's from ``edge'' option
64 %                                              by APShaw, 01 Sep 2009
65 % Modified to speed up processing by eliminating squeeze functions in
66 % the weight computations using a mixture of a squeeze and a reshape.
67 %                                              by APShaw, 08 Sep 2009
68 % Modified to add ``clutter'' specific option  by APShaw, 08 Sep 2009
69 % Clean up and improve comments to form version for ARCS study
70 % distribution and set the coding conform to MATLAB 2009b. Eliminate
71 % un-used SampleRegion options. Replace fixed offsets with
72 % parameterised values (DPCA_MASKED and LARGE_TARGET_THRESHOLD).
73 % Changed ``ClutterExtent'' variable to enable the clutter extent
74 % region to be tricked into other regions or to limited regions of
75 % the clutter. Reformated to suit incorporation into tech note.
76 %                                              by APShaw, 15 Jun 2010
77 %
78 %
79 %
80 % (c) Copyright, Commonwealth of Australia,  2009, 2010
81
82
83 %%-------------------------------------------------------------------------
84 % Fixed parameters and early initialisations
85 %-------------------------------------------------------------------------
86
87 % arbitrary scalar applied to noise cancellation
88 ARBITARY_SCALAR=1;
89
90 % Set the threshold to be used in determining if there is a target
91 % present in the training data that needs to be excised. Value is
92 % the SNR (linear)
93 LARGE_TARGET_THRESHOLD=20;
94
95 % Set the number of bins to be eliminated from the Doppler region due
96 % to the effects of DPCA
97 DPCA_MASKED=3;
98
99 % Gap in range data to be used at center of each stride, to prevent
100 % training on "self data".
101 RANGE_GAP=RangeGap;
102
```

```
103 %  Identify the number of signal and the range/Doppler map size
104 [nS nRs nDs]=size(Signal);
105
106 %  Identify the number of auxiliaries
107 [nA nRa nDa]=size(Auxiliaries);
108
109 % Initialise the output array
110 output=zeros(size(Signal))+1j*zeros(size(Signal));
111
112
113 %%-------------------------------------------------------------------------
114 % Validate the input data
115 %-------------------------------------------------------------------------
116
117 % determine if we are using pre-computed weights and validate the data
118 % provided.
119 if ~isempty(InWeights)
120     % check that the number of weight channels equals the number of aux
121     % channels
122     if ndims(InWeights)==3
123         [nWs , ~, nWc]=size(InWeights);
124         assert(and(nWs == nS, nWc == nA),...
125          'Number of weight channels must match SIG and AUX channels');
126     else
127         [~, nWc]=size(InWeights);
128         assert(nWc==nA,...
129          'Number of weight channels MUST match number of AUX channels')
130         nWs =1;
131         if nS>1
132             % if there are more signal channels than weights assume
133             % applies only to the first channel.
134             Signal(2:end,:,:)=[];
135         end
136     end
137     Nstride=nWs;
138     COMPUTEWEIGHTS=false;
139 else
140     COMPUTEWEIGHTS=true;
141 end
142
143
144 % Check number of signal/auxiliary Doppler s and ranges match
145 assert( and(nRa == nRs, nDa == nDs), ...
146     'Range and Doppler bin size of signal and auxiliaries MUST match');
147
148
149 % Check the ClutterExtent values
150 if ~isempty(ClutterExtentIn)
151     if length(ClutterExtentIn)==2
152         ClutterExtent=[1 ClutterExtentIn(1) ClutterExtendIn(2) nDs];
153     elseif length(ClutterExtentIn)>=4
154         ClutterExtent=ClutterExtentIn;
155     else
```

```
156             disp('Clutter extent badly defined - using default region');
157             SampleRegion='default';
158             ClutterExtent=[1 nDs nDs nDs];
159         end
160         ClutterExtent=sort(ClutterExtent);
161     else
162         disp('Clutter extent is not defined - using default region');
163         SampleRegion='default';
164         ClutterExtent=[1 nDs nDs nDs];
165     end
166
167     % Set the values for the step sizes in range and Doppler depending
168     % upon the value(s) passed by Nstep.
169     if length(Nstep)==1
170         DNranges=max(Nstep,1);
171         DNdopplers=max(Nstep,1);
172     else
173         DNranges=max(Nstep(1),1);
174         DNdopplers=max(Nstep(2),1);
175     end
176     DNranges_half=floor(Nranges.*DNranges./2);
177     DNdopplers_half=floor(Ndopplers.*DNdopplers./2);
178
179
180     % Check the number of ranges/dopplers is within the size of the arrays
181     assert( ((Nranges.*DNranges))<= nRa,...
182         'Value of <Nranges*Nstep> larger than number range bins in data');
183     assert( Ndopplers.*DNdopplers<= nDa,...
184         'Value of <Ndopplers*Nstep> larger than doppler bins in data');
185
186     %%-------------------------------------------------------------------------
187     % Determine regions to be used for estimations
188     %-------------------------------------------------------------------------
189
190     % compute the center range bin index for each stride
191     start=DNranges_half+1;
192     stop=max(nRa-DNranges_half,start);
193     check=[];
194     if Nstride>1
195         strides=floor(start+(stop-start)*(0:Nstride-1)/(Nstride-1));
196         % remove any strides with the same value (to prevent redundant
197         % calculations and interpolation failing)
198         check_count=0;
199         for count=2:Nstride
200             if strides(count)==strides(count-1)
201                 check_count=check_count+1;
202                 check(check_count)=count; %#ok<AGROW>
203             end
204         end
205         if check_count
206             strides(check)=[];
207         end
208     else
```

```
209     strides=floor(nRa./2);
210 end
211 Nstride=length(strides);
212
213 % compute the centres of the Doppler samples and generate a list of the
214 % Doppler bins to be used for weight estimation.
215 Doppler_step=max(1,DNdopplers);
216 SampleRegion=lower(SampleRegion);
217 switch SampleRegion
218     case 'clutter'
219         % Make a list equal to the entire Doppler space
220         Doppler_list0=1:Doppler_step:nDs;
221         % Eliminate those bins that are NOT in the clutter region
222         Doppler_list0(...
223             and(Doppler_list0 > ClutterExtent(2), ...
224                 Doppler_list0 < ClutterExtent(3)))=[];
225         Doppler_list0(Doppler_list0 < ClutterExtent(1))=[];
226         Doppler_list0(Doppler_list0 > ClutterExtent(4))=[];
227         % Use the eliminated region as the Doppler list for training
228         Doppler_list=Doppler_list0;
229         % remove the DPCA_masked region from the training list
230         Doppler_list(Doppler_list<=DPCA_MASKED)=[];
231         Doppler_list(Doppler_list>=nDs-DPCA_MASKED)=[];
232     case 'noise'
233         % Make a list equal to the entire Doppler space
234         Doppler_list0=1:Doppler_step:nDs;
235         % Eliminate those bins that are in the clutter region
236         Doppler_list0(Doppler_list0<= ClutterExtent(2))=[];
237         Doppler_list0(Doppler_list0>= ClutterExtent(3))=[];
238
239         % Use the eliminated region as the Doppler list for training
240         Doppler_list=Doppler_list0;
241     otherwise %'default'
242         D0=floor(nDs./2);
243         Doppler_list=...
244             max(1,D0-DNdopplers_half): ...
245                 Doppler_step:min(D0+DNdopplers_half,nDa);
246 end
247 NumberDopplers=length(Doppler_list);
248
249 % Failsafe provision - ensure we have at least one Doppler in the list
250 if NumberDopplers<1
251     Doppler_list=floor(nDa./2);
252     NumberDopplers=1;
253 end
254
255
256 %%-------------------------------------------------------------------------
257 % Estimate the weight sets for each stride
258 %-------------------------------------------------------------------------
259
260 if COMPUTEWEIGHTS
261     % initialise the weights and the intermediate matrices
```

```
262     weights=zeros(Nstride,nA,nS)+1j.*zeros(Nstride,nA,nS);
263     M_init=zeros(nA)+1j.*zeros(nA);
264     R_init=zeros(nS,nA)+1j.*zeros(nS,nA);
265     total_power=ones(nS,1);
266
267     % loop over the number of range strides to take over the data set
268     for stride_counter=1:Nstride
269
270         % Determine the extent of region where range data is to be
271         % taken, making accomodation for the gap in the range data
272         % and retaining the specified number of range bins in the
273         % data set.
274         Range1=strides(stride_counter)-DNranges_half-RANGE_GAP;
275         Range2=strides(stride_counter)+DNranges_half+RANGE_GAP;
276
277         % If the range extents go beyond the limits of valid data,
278         % adjust them to be within the range limits
279         if Range2>nRa
280             Excess=Range2-nRa;
281             Range1=Range1-Excess;
282             Range2=nRa;
283         end
284         if Range1<1
285             Excess=1-Range1;
286             Range2=Range2+Excess;
287             Range1=1;
288         end
289         % Double check the effects of the offsets incase a large
290         % number of range bins are in use.
291         Range1=max(1,Range1);
292         Range2=min(Range2,nRa);
293
294         % Determine the extent of the region where there is a gap
295         % in the range data.
296         Skip1=strides(stride_counter)-RANGE_GAP;
297         Skip2=strides(stride_counter)+RANGE_GAP;
298
299         % Check the guard region doesn't go beyond the limits of valid
300         % data and if they do adjust them accordingly
301         if Skip2>nRa
302             Excess=Skip2-nRa;
303             Skip1=Skip1-Excess;
304             Skip2=nRa;
305         end
306         if Skip1<1
307             Excess=1-Skip1;
308             Skip2=Skip2+Excess;
309             Skip1=1;
310         end
311         % Double check in case a large value of RANGE_GAP is in use.
312         Skip1=max(1,Skip1);
313         Skip2=min(Skip2,nRa);
314
```

```
315         % Make a list of the range bins to be used for this stride and
316         % then remove those that are in the skip region
317         ValidRanges=Range1:DNranges:Range2;
318         ValidRanges(and(ValidRanges>=Skip1,ValidRanges<=Skip2))=[];
319
320         % Fail-safe provision: ensure that we have at least one range
321         % bin left in the stride.
322         if length(ValidRanges)<1
323             ValidRanges=strides(stride_counter);
324         end
325
326
327         % generate a vector of total signal powers in this stride
328         % accomodating the gap.
329         for counter=1:nS
330             total_power(counter,:)=squeeze(...
331                 sum(sum(Signal(nS,ValidRanges,Doppler_list) ...
332                 .*conj(Signal(nS,ValidRanges,Doppler_list)))));
333         end
334         cells=squeeze((length(ValidRanges).*length(Doppler_list))-1);
335
336         % initialise weight computation
337         M=M_init;
338         R=R_init;
339
340         % compute the covariance matrix for this range stride
341         for range_counter=1:length(ValidRanges)
342             range_bin=ValidRanges(range_counter);
343
344             % Speed up functionality: extract data for this range bin
345             T_Signal=squeeze(Signal(:,range_bin,:));
346             if nS==1
347                 T_Signal=T_Signal.';
348             end
349             T_Auxiliaries=squeeze(Auxiliaries(:,range_bin,:));
350
351             % speed up functionality: reshape to avoid "squeeze" inside
352             % the inner loop.
353             t_signal=reshape(T_Signal,size(T_Signal,1) ...
354                 *size(T_Signal,2),1);
355             t_Auxiliaries=reshape(T_Auxiliaries,...
356                size(T_Auxiliaries,1)*size(T_Auxiliaries,2),1);
357
358              for doppler_counter=1:NumberDopplers
359                 doppler_bin=Doppler_list(doppler_counter);
360                 Svector=t_signal((doppler_bin-1)*nS+1: ...
361                     (doppler_bin-1)*nS+nS);
362                 Spower=Svector.*conj(Svector);
363
364                 % If there isn't a large target present in the test
365                 % cell then add this cell
366                 if ~(max((Spower./((total_power-Spower)./cells)...
367                     >LARGE_TARGET_THRESHOLD)))
```

```
368                         Avector=t_Auxiliaries((doppler_bin-1)*nA+1: ...
369                             (doppler_bin-1)*nA+nA);
370                         M=M + (Avector)*Avector';
371                         R=R + (Svector)*conj(Avector).';
372                     %                   counter=counter+1;
373                     end
374                 end
375             end
376
377             %Normalisations turned off to save computational cycles
378             %M=M./counter;
379             %R=R./counter;
380
381             % Compute the weights: note MATLAB recommendation to not
382             % use inv function.
383             % weights(stride_counter,:,:)=inv(M)*R';
384             weights(stride_counter,:,:)=M\R';
385         end
386         % need to take complex conjugate of the weights - this is because
387         % we have allowed MATLAB to do the conjugate transpose instead of
388         % the transpose as this is marginally faster on the test machine.
389         weights=conj(weights);
390     else
391         % If we are not computing the weights from the data, then use the
392         % weights that have been supplied in the input parameters.
393         weights=InWeights;
394     end
395
396
397     %%-------------------------------------------------------------------------
398     % Apply the weighted auxiliaries to the signal
399     %-------------------------------------------------------------------------
400
401     % Range values, using "start" and "stop" (calculated previously) values
402     % to avoid extrapolation
403     ranges=min(max(1:nRs,start),stop);
404     % the Doppler expansion of the range weights to simplify the weight
405     % application
406     expander=ones(nDs,1);
407     % initialise the correction matrix
408     correction0=zeros(nRs,nDs)+1j*zeros(nRs,nDs);
409     % initialise the output matrix
410     %output=zeros(nS,nRs,nDs)+1j*zeros(nS,nRs,nDs);
411
412     % Apply the weights to each signal in turn
413     for counter0=1:nS
414
415         %extract the weights for the current signal
416         Sweights=squeeze(weights(:,:,counter0));
417
418         % initialise the correction for this signal channel
419         correction=correction0;
420
```

```
421     %compute the weight to be applied at each range bin (output weights
422     %array has dimensions nA x nRs
423     if Nstride==1
424         % one weight applies to every range bin
425         Wout=((squeeze(Sweights(1,:))).')*ones(1,nRs);
426
427         % compute the correction factor
428         for counter=1:nA
429             correction=correction+ ...
430                 squeeze(Auxiliaries(counter,:,:)).* ...
431                 ((expander*Wout(counter,:))).');
432         end
433     else
434         % Different weights for each range bin
435         for counter=1:nA
436             % interpolate the computed strides to each range bin for
437             % each array for this signal channel
438             Win=squeeze(Sweights(:,counter));
439             Wout(counter,:)=interp1(strides,Win,ranges,'spline');
440
441             % compute the correction factor
442             correction=correction+ ...
443                 squeeze(Auxiliaries(counter,:,:)).* ...
444                 ((expander*Wout(counter,:))).');
445         end
446     end
447
448     % apply the correction to the sigal, including the arbitrary scalar
449     % adjustment of the weighted correction.
450     switch SampleRegion
451         case 'clutter'
452             % Initially make the output equal to the input
453             output(counter0,:,:)=squeeze(Signal(counter0,:,:));
454             % Apply the correction only over the entire clutter region
455             % Note - this does include the DPCA_MASKED region.
456             output(counter0,:,Doppler_list0)=...
457                 squeeze(Signal(counter0,:,Doppler_list0))-...
458                 correction(:,Doppler_list0)*ARBITARY_SCALAR;
459         otherwise
460             output(counter0,:,:)=squeeze(Signal(counter0,:,:))-...
461                 correction.*ARBITARY_SCALAR;
462     end
463 end
```

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. CAVEAT/PRIVACY MARKING
2. TITLE An Implementation of Adaptive Side Lobe Cancellation in MATLAB®			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)	
4. AUTHORS A. P. Shaw			5. CORPORATE AUTHOR Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia	
6a. DSTO NUMBER DSTO–TN–0955	6b. AR NUMBER 014–078		6c. TYPE OF REPORT Technical Note	7. DOCUMENT DATE July, 2010
8. FILE NUMBER 2010/1076861/1	9. TASK NUMBER 07/044	10. SPONSOR PM C & W	11. No OF PAGES 14	12. No OF REFS 1
13. URL OF ELECTRONIC VERSION http://www.dsto.defence.gov.au/corporate/reports/DSTO–TN–0955.pdf			14. RELEASE AUTHORITY Chief, Electronic Warfare and Radar Division	
15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT *Approved For Public Release* OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111				
16. DELIBERATE ANNOUNCEMENT No Limitations				
17. CITATION IN OTHER DOCUMENTS No Limitations				
18. DSTO RESEARCH LIBRARY THESAURUS Radar, Radar interference, Radar jamming, Signal Processing, Adaptive signal processing, Side lobes, Software, Computer programs				
19. ABSTRACT This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.				